

Results from the *single-read* test

Adam Lyon, August 2, 2005

1 Introduction

A SAM test was performed to determine the rate of file deliveries possible on a test CAF system.

Appendix A R Session

R is an open source statistical analysis software package that allows for very easy analysis of data in databases and text files. I wrote a "notebook" style package that allows one to use R from within Microsoft Word. Below is the notebook providing all of the code and results for this document.

A.1 Connect to R and initialize

Connect to R running locally on my laptop.

```
<R0> #connect port 6101 timeout 20
R is using work directory /Users/adam/work/projects/cdfSamTests/single-
read
```

Set up graphics

```
<R1> library(lattice)
```

```
<R2> trellis.par.set(col.whitebg())
```

```
<R3> fontsize = trellis.par.get("fontsize"); fontsize$text=16 ;
      fontsize$points=6 ; trellis.par.set("fontsize", fontsize)
```

I have a helper function that makes putting graphics into Word easy.

```
<R4> mp
function (plotExpr, file, height = 7, width = 7, res = 72 * 3)
{
  bitmap(file, "pngalpha", height = height, width = width,
         res = res, pointsize = 10)
  r = eval(plotExpr)
  if (class(r) == "trellis")
    print(r)
  invisible(dev.off())
}
<environment: namespace:RemoteRSOAP>
```

A.2 Running job output

Doug's python script loops over getting the next file, waiting thirty seconds to simulate processing, and then releasing the file. A log file records the time of the get next file, the duration of the command, the pnfs name of the file, and the time and duration of the release file command.

Let's read this information into R.

```

<R5> d = read.table("out.log", header=T)

<R6> d[1:5,]
   job segment   getDate   getTime getDur fileNum
1 2365        1 2005-08-02 20:52:32  1.407      1
2 2365        1 2005-08-02 20:52:48  1.066      2
3 2365        1 2005-08-02 20:53:04  1.067      3
4 2365        1 2005-08-02 20:53:21  1.070      4
5 2365        1 2005-08-02 20:53:37  1.072      5

pnfs
1
dcap://cdfdca1.fnal.gov:25138/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ23/
GJ2308/GJ2308.0/xd025e0c.0100bhd0
2
dcap://cdfdca1.fnal.gov:25138/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ23/
GJ2314/GJ2314.0/xd025eb8.0041bhd0
3
dcap://cdfdca1.fnal.gov:25138/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ23/
GJ2321/GJ2321.0/xd025df9.01c9bhd0
4
dcap://cdfdca1.fnal.gov:25138/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ23/
GJ2320/GJ2320.0/xd025e09.009dbhd0
5
dcap://cdfdca1.fnal.gov:25138/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ23/
GJ2327/GJ2327.0/xd025ed1.0155bhd0
   relDate  relTime relDur
1 2005-08-02 20:52:48  0.248
2 2005-08-02 20:53:04  0.114
3 2005-08-02 20:53:20  0.112
4 2005-08-02 20:53:37  0.091
5 2005-08-02 20:53:53  0.109

```

Convert dates and times into POSIX times that R can deal with...

```

<R7> d$getDateTime = paste(d$getDate, d$getTime)

<R8> d$relDateTime = paste(d$relDate, d$relTime)

<R9> d$getPTime = as.POSIXct( strptime( d$getDateTime, "%Y-%m-%d
%H:%M:%S" ) )

<R10> d$relPTime = as.POSIXct( strptime( d$relDateTime, "%Y-%m-%d
%H:%M:%S" ) )

```

We can make the delivery time from the get time plus the duration

```
<R11> d$delPTime = d$getPTime + d$getDur
```

What is the range of the test?

```
<R12> testEdges = c(min(d$getPTime), max(d$relPTime, na.rm=T)) ;  
      testEdges  
[1] "2005-08-02 20:52:32 CDT" "2005-08-03 06:36:05 CDT"  
  
<R13> testTime = diff(testEdges) ; testTime  
Time difference of 9.725833 hours  
  
<R14> prettyEdges = c(testEdges[1]-60*60, testEdges[2]+60*60)
```

A.2.1 Basics

How many files deliveries were attempted?

```
<R15> nrow(d)  
[1] 2000
```

How many failed on the get end?

```
<R16> sum(is.na(d$getDur))  
[1] 0
```

How many failed on the release end?

```
<R17> sum(is.na(d$relDur))  
[1] 0
```

Merge job and segment numbers

```
<R18> d$jobseg = paste(d$job, d$segment)
```

What were the jobs and segments?

```
<R19> d$jobseg[ is.na(d$relDur) ]  
character(0)  
  
<R20> #var successfulDeliveries = nrow(d)  
2000
```

All files were delivered!

Look up the maximum file number for each job and segment.

```
<R21> largestFNum = tapply(d$fileNum, d$jobseg, max)
```

How many did not get all 40 files?

```
<R22> length( largestFNum[ largestFNum < 40 ] )  
[1] 0
```

Hmmm.

```
<R23> notDone = largestFNum[ largestFNum < 40 ]
```

```
<R24> notDone[1:3]  
<NA> <NA> <NA>  
NA NA NA
```

A.2.2 File delivery rate

```
<R25> fileRatePerDay = nrow(d)/(as.numeric(testTime))*24;  
      fileRatePerDay  
[1] 4935.31
```

In a perfect world, what should be the mean wait time?

```
<R26> filesPerSeg = nrow(d) / length(unique(d$jobseg)) ; filesPerSeg  
[1] 2000
```

Assume that every segment runs the entire length of the test. Therefore (in minutes),

```
<R27> meanWaitTime = as.numeric(testTime)*60 / filesPerSeg ;  
      meanWaitTime  
[1] 0.291775
```

How many seconds per file?

```
<R28> secondsPerFile = as.numeric(testTime)*60*60 / nrow(d) ;  
      secondsPerFile  
[1] 17.5065
```

A.2.3 Number of segments running

Let's plot how many segments were running at a given time.

A segment turns on when it does its first "get next file". It turns off when it does its last release.

How many segments are there?

```
<R29> length( unique( d$jobseg) )  
[1] 1
```

So we have a record of all 1000 segments. Good!

Segment starts when the first file is requested

```
<R30> segStart = data.frame( time=d$getPTime[d$fileNum==1], adj=1 )
```

```

<R38> segEnd = data.frame( time=d$relPTime[d$fileNum==2000], adj=-1 )

<R39> segs = rbind(segStart, segEnd)

<R40> segs = segs[ order(segs$time), ]

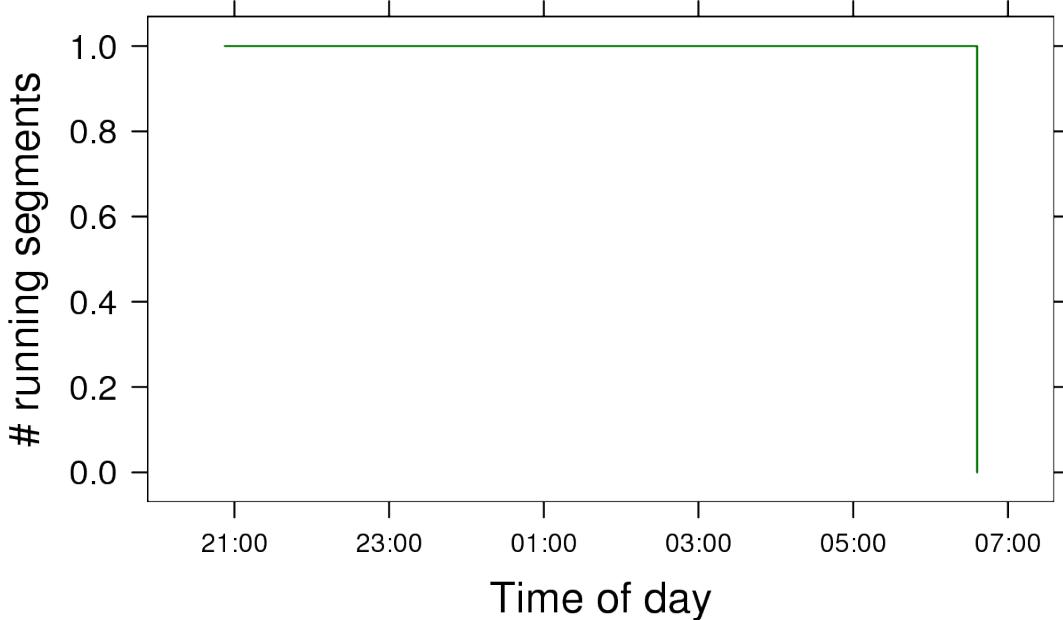
<R41> segs$count = cumsum(segs$adj)

<R42> segs[1:2,]
      time adj count
1 2005-08-02 20:52:32   1     1
11 2005-08-03 06:36:05  -1     0

<R43> mp(
  xyplot( segs$count ~ segs$time, type="s",
          main="Number of running segments",
          xlab="Time of day",
          ylab="# running segments",
          scales=list(x=list(tick.number=6, cex=0.6)),
          xlim=prettyEdges ),
  "nsegs.png", h=4, w=6 )
#with graphics nsegs.png timeout 120

```

Number of running segments



A.2.4 Number of gets and deliveries

Let's count up how many get file requests and deliveries there were per hour

```
<R44> getsPerHourCuts = cut( d$getPTime, "hours" )

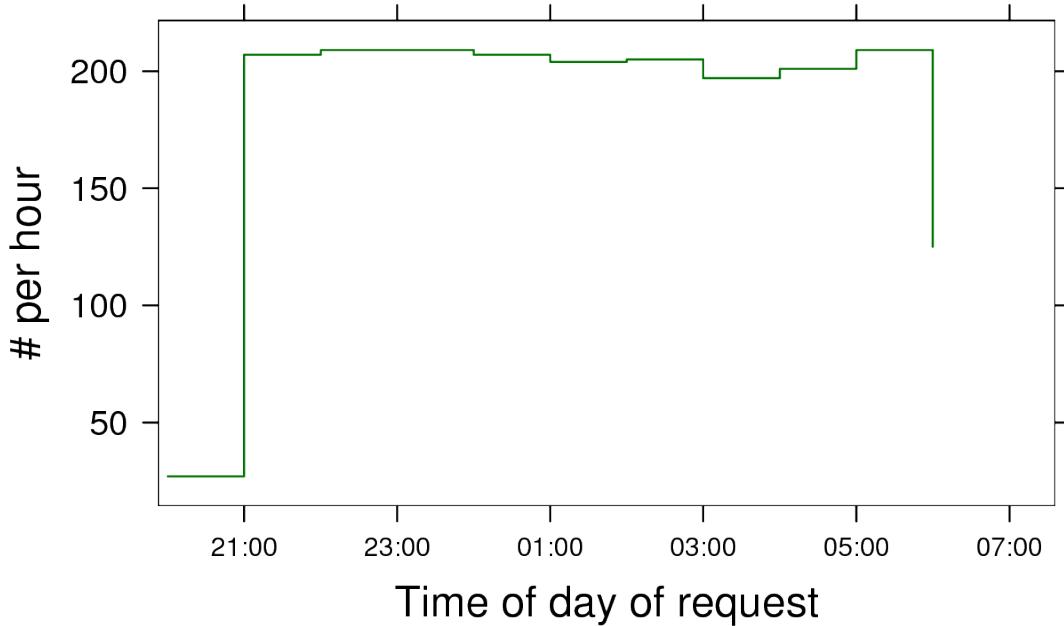
<R45> delsPerHourCuts = cut( d$delPTime, "hours" )

<R46> getsPerHour = tabulate(getsPerHourCuts)

<R47> delsPerHour = tabulate(delsPerHourCuts)

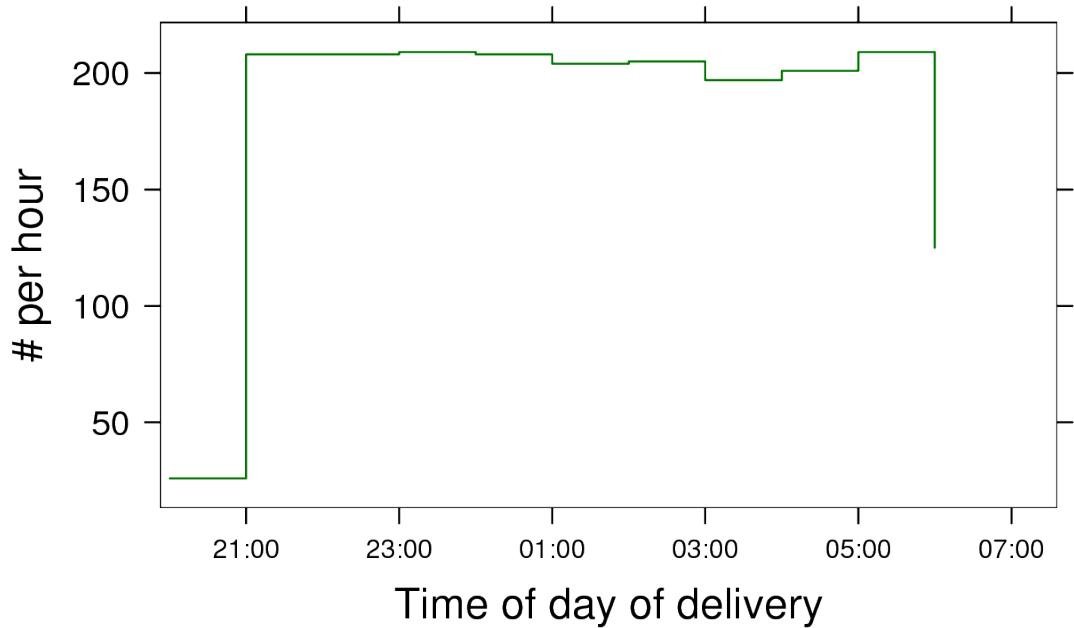
<R48> mp(
  xyplot( getsPerHour ~ as.POSIXct(levels(getsPerHourCuts)),
    main="Get file requests",
    xlab="Time of day of request",
    ylab="# per hour", type="s", xlim=prettyEdges,
    scales=list(x=list(tick.number=6, cex=0.6)))
),
"getsPerHour.png", h=4, w=6
)
#with graphics getsPerHour.png timeout 60
```

Get file requests



```
<R49> mp(
  xyplot( delsPerHour ~ as.POSIXct(levels(delsPerHourCuts)),
    main="File deliveries",
    xlab="Time of day of delivery", xlim=prettyEdges,
    ylab="# per hour", type="s",
    scales=list(x=list(tick.number=6, cex=0.6))
  ),
  "delsPerHour.png", h=4, w=6
)
#with graphics delsPerHour.png timeout 60
```

File deliveries



Let's just get a cumulative plot of when deliveries occurred.

```
<R50> deliveries = data.frame( time=d$delPTime, adj=1 )
```

```
<R51> deliveries = deliveries[ order(deliveries$time), ]
```

```
<R52> deliveries$count = cumsum(deliveries$adj)
```

```
<R53> nrow(deliveries)
```

```
[1] 2000
```

```
<R54> deliveries[39270:39281,]
```

```
  time adj count
```

```
NA     <NA>  NA   NA
```

```
NA.1    <NA>  NA   NA
```

```
NA.2    <NA>  NA   NA
```

```
NA.3    <NA>  NA   NA
```

```
NA.4    <NA>  NA   NA
```

```
NA.5    <NA>  NA   NA
```

```
NA.6    <NA>  NA   NA
```

```
NA.7    <NA>  NA   NA
```

```
NA.8    <NA>  NA   NA
```

```
NA.9    <NA>  NA   NA
```

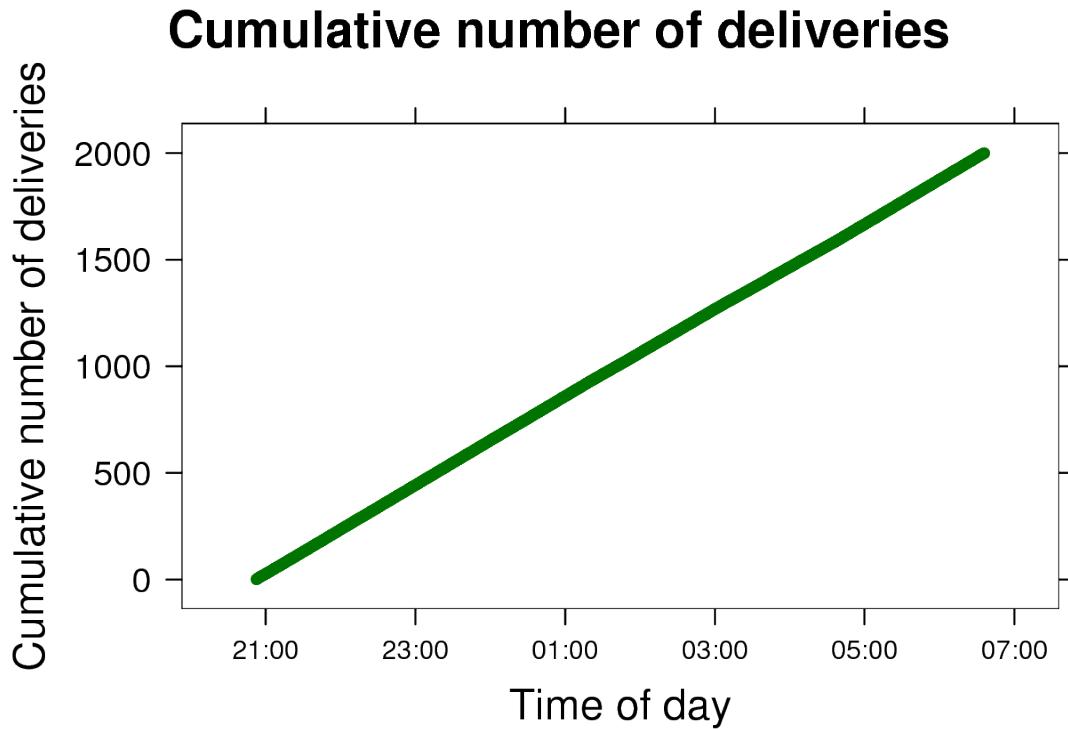
```
NA.10   <NA>  NA   NA
```

```
NA.11   <NA>  NA   NA
```

```

<R55> mp(
  xyplot( deliveries$count ~ deliveries$time, type="p",
         main="Cumulative number of deliveries",
         xlab="Time of day",
         ylab="Cumulative number of deliveries",
         scales=list(x=list(tick.number=6, cex=0.6)),
         xlim=prettyEdges ),
  "ndel.png", h=4, w=6 )
#with graphics ndel.png timeout 120

```



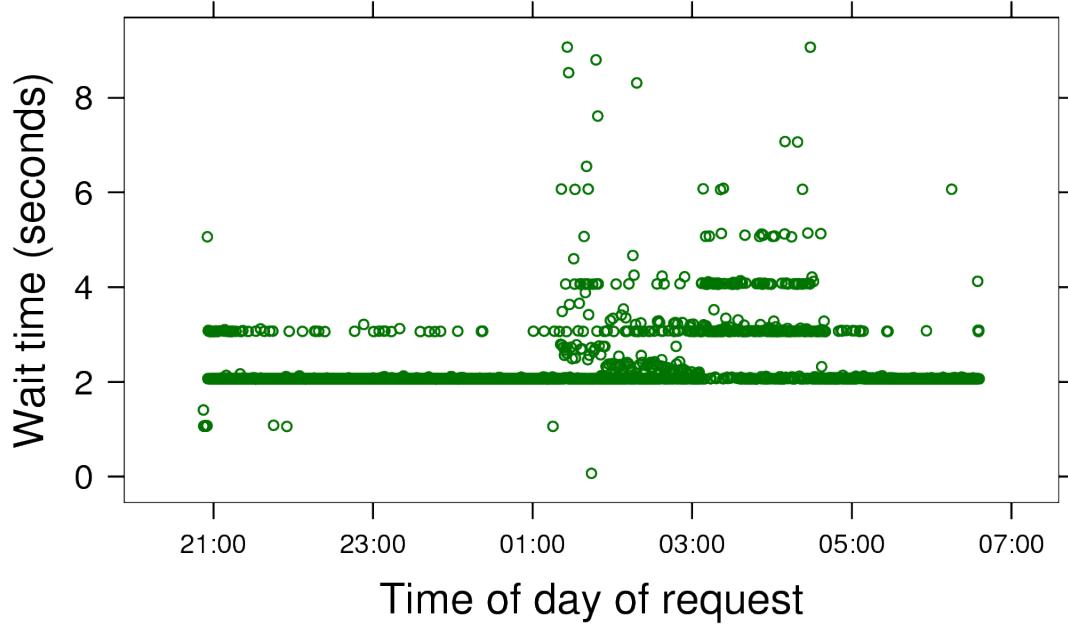
What do the wait times look like?

```

<R57> mp(
  xyplot( getDur ~ getPTime, data=d,
         main="File delivery waits",
         xlab="Time of day of request", xlim=prettyEdges,
         ylab="Wait time (seconds)",
         scales=list(x=list(tick.number=6, cex=0.6)))
  ),
  "getWaits.png", h=4, w=6
)
#with graphics getWaits.png timeout 60

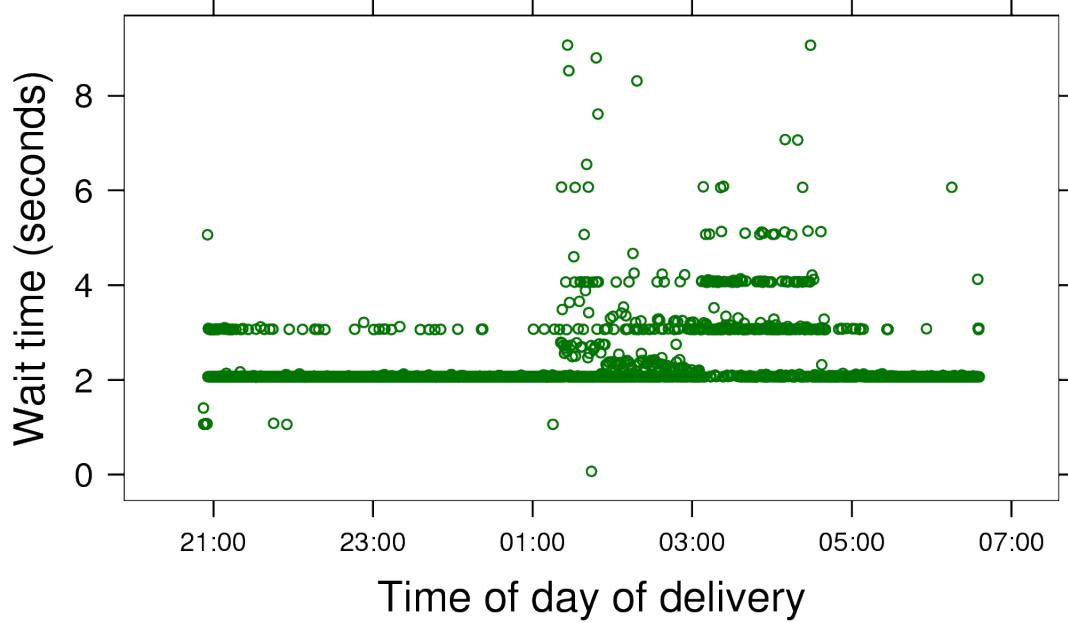
```

File delivery waits



```
<R58> mpC
xyplot( getDur ~ delPTime, data=d,
        main="File delivery waits",
        xlab="Time of day of delivery", xlim=prettyEdges,
        ylab="Wait time (seconds)",
        scales=list(x=list(tick.number=6, cex=0.6))
      ),
"delWaits.png", h=4, w=6
)
#with graphics delWaits.png timeout 60
```

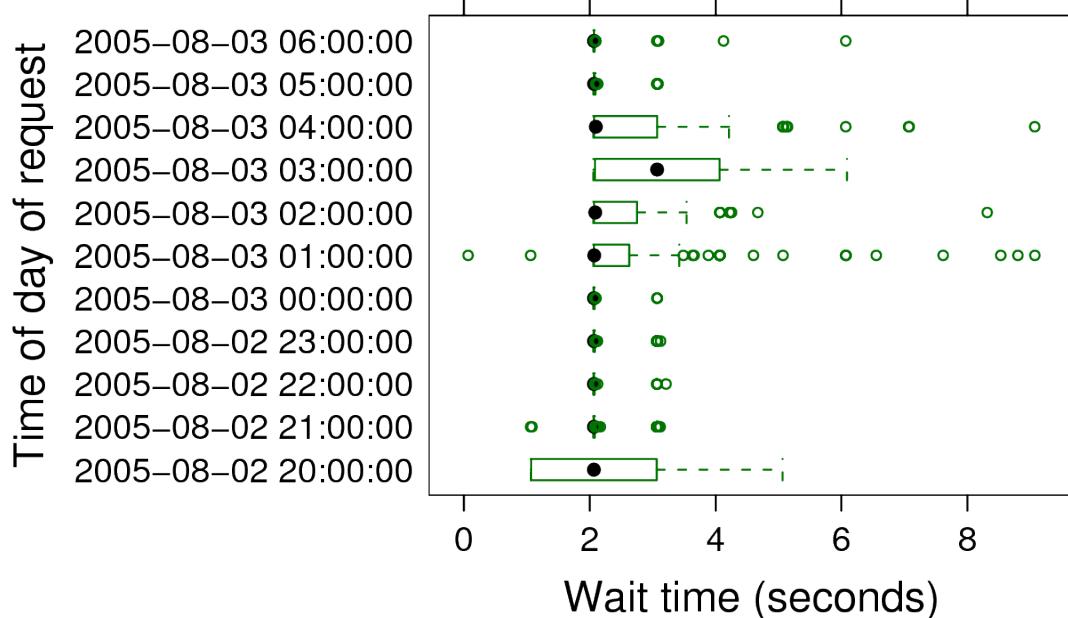
File delivery waits



These plots make the outliers really stand out and hard to see the mean wait times. Let's try to plot those...

```
<R59> mp(
  bwplot( getsPerHourCuts ~ getDur, data=d,
    main="File delivery waits",
    xlab="Wait time (seconds)", ylab="Time of day of request"
  ),
  "waitsPerHourBw.png", h=4, w=6
)
#with graphics waitsPerHourBw.png timeout 60
```

File delivery waits



Again, the outliers dominate the plot. Let's just plot medians and how many are over an hour...

```
<R60> waitMedians = tapply(d$getDur, getsPerHourCuts, median)

<R61> waitMeans = tapply(d$getDur, getsPerHourCuts, mean)

<R62> nOverHour = tapply(d$getDur >= 4, getsPerHourCuts, sum)

<R63> percOverHour = nOverHour / getsPerHour * 100

<R64> nOverHalfHour = tapply(d$getDur >= 3, getsPerHourCuts, sum)

<R65> percOverHalfHour = nOverHalfHour / getsPerHour * 100

<R66> nUnderMinute = tapply(d$getDur < 1, getsPerHourCuts, sum)

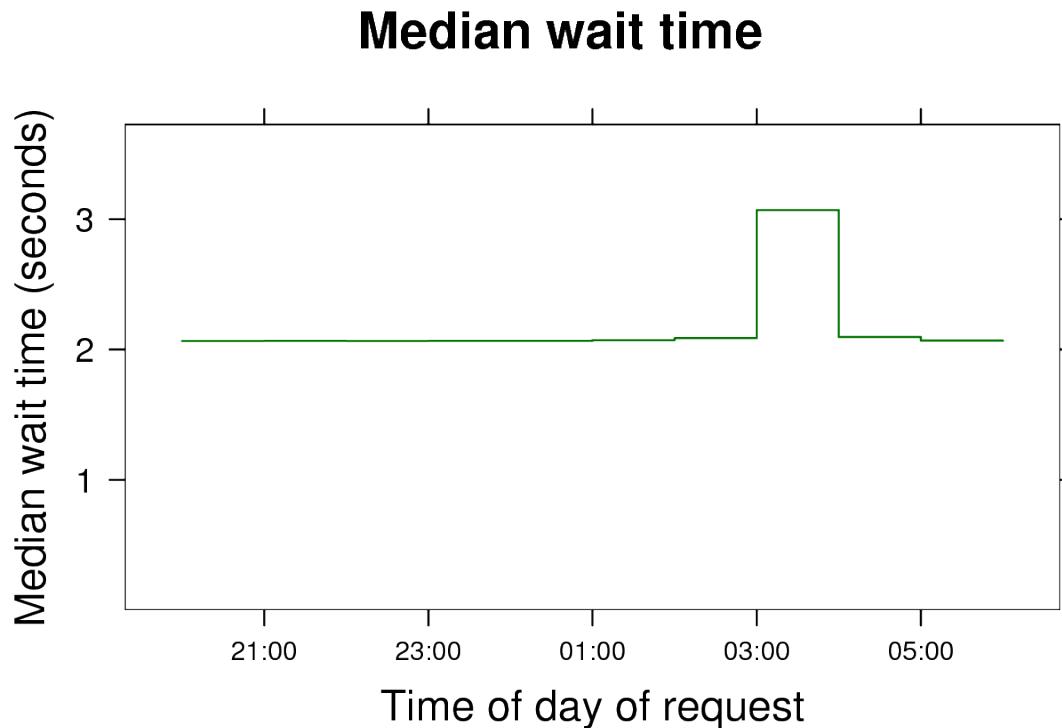
<R67> percUnderMinute = nUnderMinute / getsPerHour * 100
```

Let's plot these things

```

<R68> mp(
  xyplot( waitMedians ~ as.POSIXct(names(waitMedians)), type="s",
         main="Median wait time", xlab="Time of day of request",
         ylab="Median wait time (seconds)",
         ylim=c(0, max(waitMeans)*1.2),
         scales=list(x=list(tick.number=6, cex=0.6))
  ),
  "medians.png", h=4, w=6)
#with graphics medians.png timeout 60

```

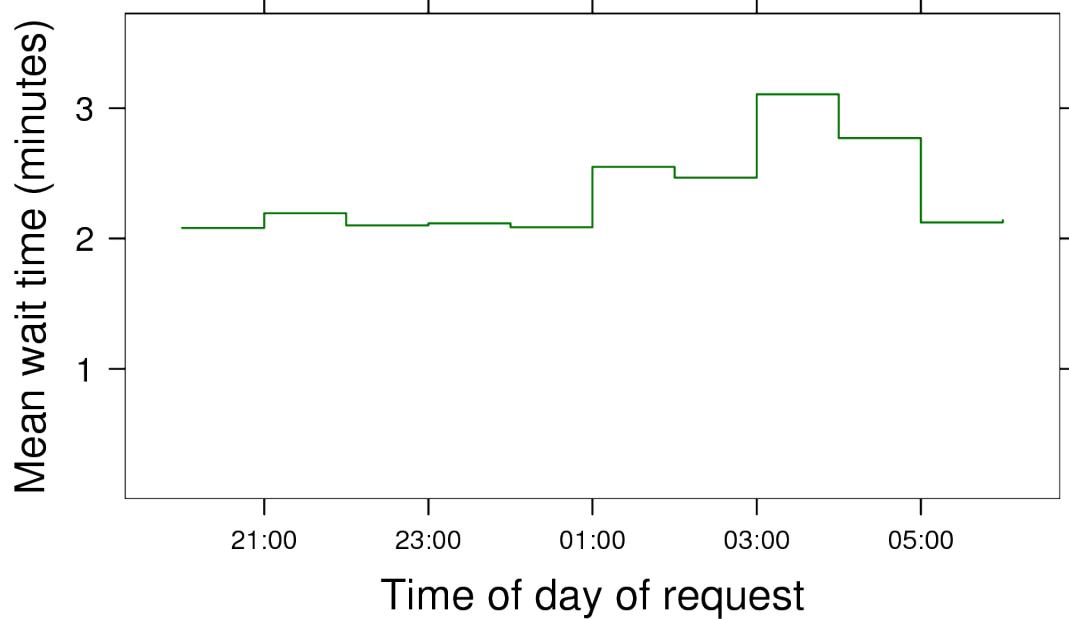


```

<R69> mp(
  xyplot( waitMeans ~ as.POSIXct(names(waitMeans)), type="s",
         main="Mean wait time", xlab="Time of day of request",
         ylab="Mean wait time (minutes)",
         ylim=c(0, max(waitMeans)*1.2),
         scales=list(x=list(tick.number=6, cex=0.6))
  ),
  "means.png", h=4, w=6)
#with graphics means.png timeout 60

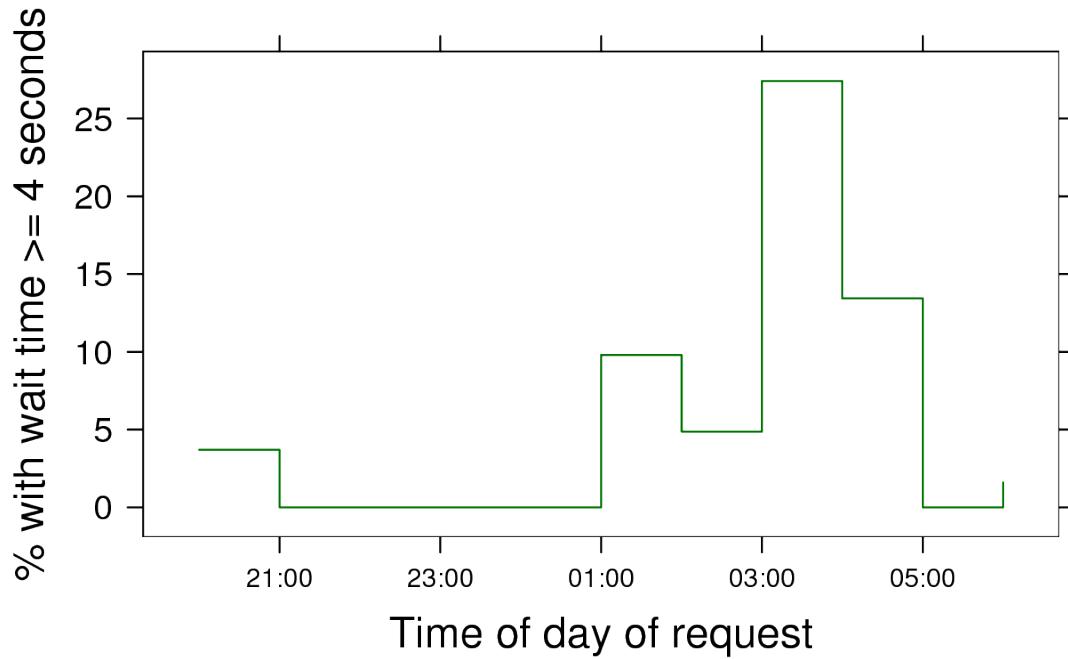
```

Mean wait time



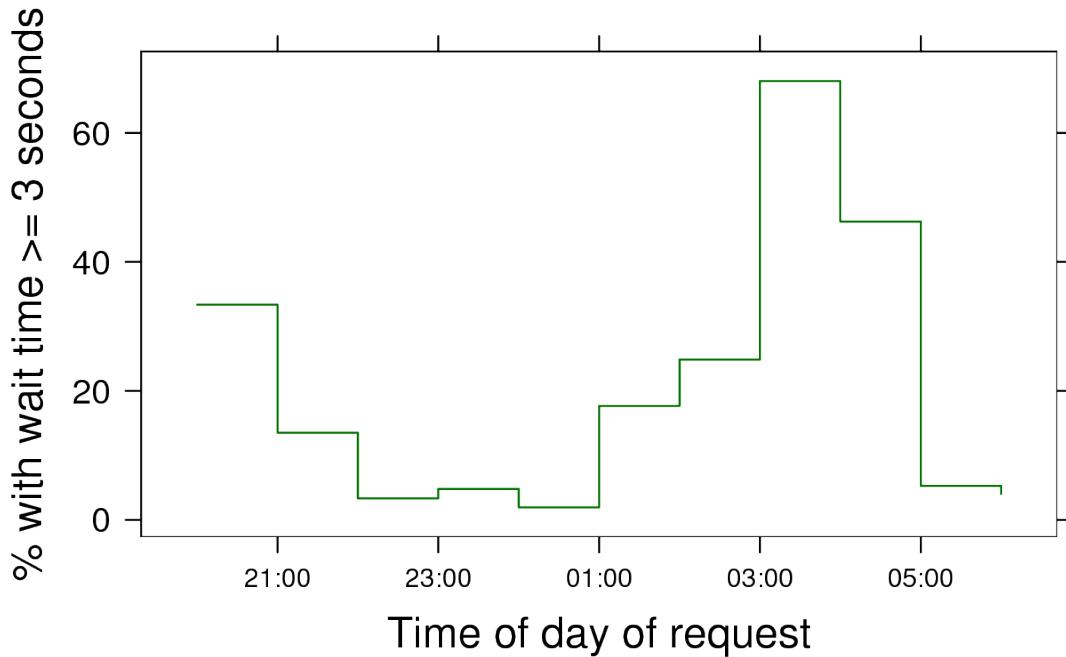
```
<R70> mp(
  xyplot( percOverHour ~ as.POSIXct(names(percOverHour)), type="s",
         main="Percent long wait times", xlab="Time of day of
request",
         ylab="% with wait time >= 4 seconds",
         scales=list(x=list(tick.number=6, cex=0.6))
  ),
  "overHour.png", h=4, w=6)
#with graphics overHour.png timeout 60
```

Percent long wait times



```
<R71> mp(
  xyplot( percOverHalfHour ~ as.POSIXct(names(percOverHalfHour)),
  type="s",
    main="Percent long wait times", xlab="Time of day of
request",
    ylab="% with wait time >= 3 seconds",
    scales=list(x=list(tick.number=6, cex=0.6))
),
"overHalfHour.png", h=4, w=6)
#with graphics overHalfHour.png timeout 60
```

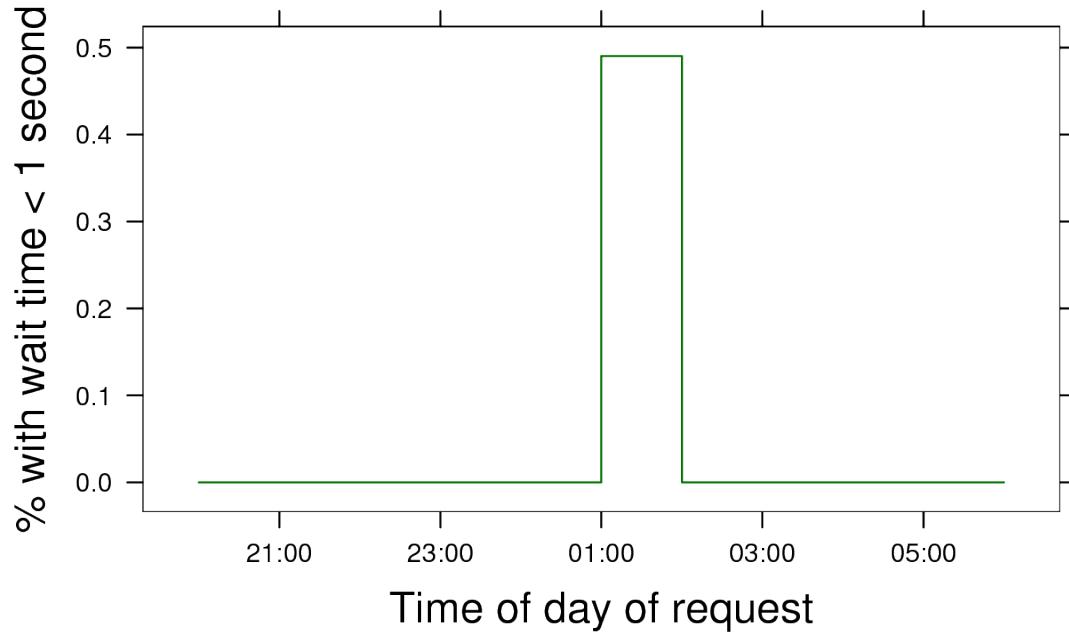
Percent long wait times



Look for short wait times.

```
<R72> mp(
  xyplot( percUnderMinute ~ as.POSIXct(names(percUnderMinute)),
  type="s",
    main="Percent short wait times", xlab="Time of day of
request",
    ylab="% with wait time < 1 second",
    scales=list(tick.number=6, cex=0.6)
),
"underMinute.png", h=4, w=6)
#with graphics underMinute.png timeout 60
```

Percent short wait times



A.2.5 Look at number of open requests

Let's look at the number of instantaneous requests that are open at any given time.

Get next file requests...

```
<R73> gnf = data.frame( time=d$getPTime, adj=1 )
```

File deliveries (request fulfilled)

```
<R74> gnc = data.frame( time=d$delPTime, adj=-1 )
```

Join them,

```
<R75> gn = rbind(gnf, gnc)
```

Sort by time,

```
<R76> gn = gn[ order(gn$time), ]
```

Do a running count of # of unfulfilled get next file requests

```
<R77> gn$count = cumsum(gn$adj)
```

```
<R78> gn[1:10,]
```

```

          time adj count
1 2005-08-02 20:52:32  1    1
11000 2005-08-02 20:52:33 -1    0
2 2005-08-02 20:52:48  1    1
2100 2005-08-02 20:52:49 -1    0
3 2005-08-02 20:53:04  1    1
3100 2005-08-02 20:53:05 -1    0
4 2005-08-02 20:53:21  1    1
4100 2005-08-02 20:53:22 -1    0
5 2005-08-02 20:53:37  1    1
5100 2005-08-02 20:53:38 -1    0

<R79> summary(gn$count)
   Min. 1st Qu. Median     Mean 3rd Qu.     Max.
   0.0    0.0    0.5    0.5    1.0    1.0

```

Plot it...

```

<R82> mp(
  xyplot( gn$count ~ gn$time, type="l",
         main="Pending get next file requests",
         xlab="Time of day",
         ylab="Instantaneous # of unfulfilled requests",
         scales=list(x=list(tick.number=6, cex=0.6)),
         xlim=prettyEdges, pex=0.1 ),
  "unfulfilled.png", h=4, w=6 )
#with graphics unfulfilled.png timeout 240

```

